What is a software process model?

- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.
- Generic software process models:

1. The waterfall model - Separate and distinct phases of specification and development.
2. Prototyping.
3. Evolutionary development - Specification and development are interleaved.
4. Formal systems development - A mathematical system model is formally transformed to an implementation.
5. Reuse-based development- The system is assembled from existing components.

Why have models?

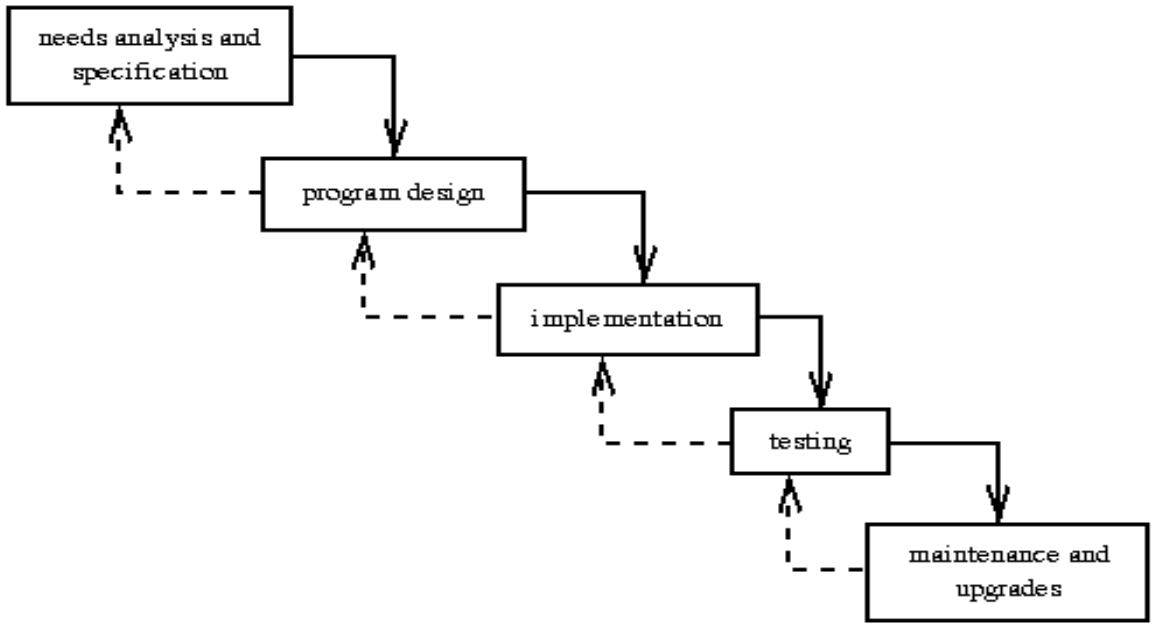
- Software has become vital in our everyday life.
- Software programs are large complicated.
- A model recognizes that Software Engineering is more than just programming.
- Recognize product life cycles.
- A model helps recognize and define the division of labor
  - Individual responsibilities (program managers, software design engineers, UI designers, testers, product support, marketing, etc.)
  - How big should a team be
  - Parallel work efforts
  - Does a one person team need a model
- Provide a common means of communication between all involved parties
- Documentation is vital- Comments in the code is not sufficient documentation

Goals of a good model:

1. The obvious "Provide a framework for building a solidly engineered product".
2. A paradigm that adds discipline and order to software development.
3. Provides a formal mechanism to clarify, track, and modify the product requirements throughout the product life cycle.
4. Provide a guideline for engineers to use in the product life cycle.
5. Provide feedback into the development cycle.

Software lifecycle:

A software engineering lifecycle model describes how one might put structure on the software engineering activities. The classic lifecycle model is the waterfall model (roughly due to Royce in 1956), which is structured by a set of activities and is inherently document-driven. The cost of fixing errors at later lifecycle phases is much higher than at earlier stages. In the lifecycle, the feedback must be increased to reduce these costs (and other risks).



"Software lifecycle"

1. Waterfall Model:

Applicability

- When the requirements are well-understood and changes will be fairly limited during the design process.
- It is mostly used for large systems engineering projects where a system is developed at several sites.

It consists of:

Requirements analysis and definition

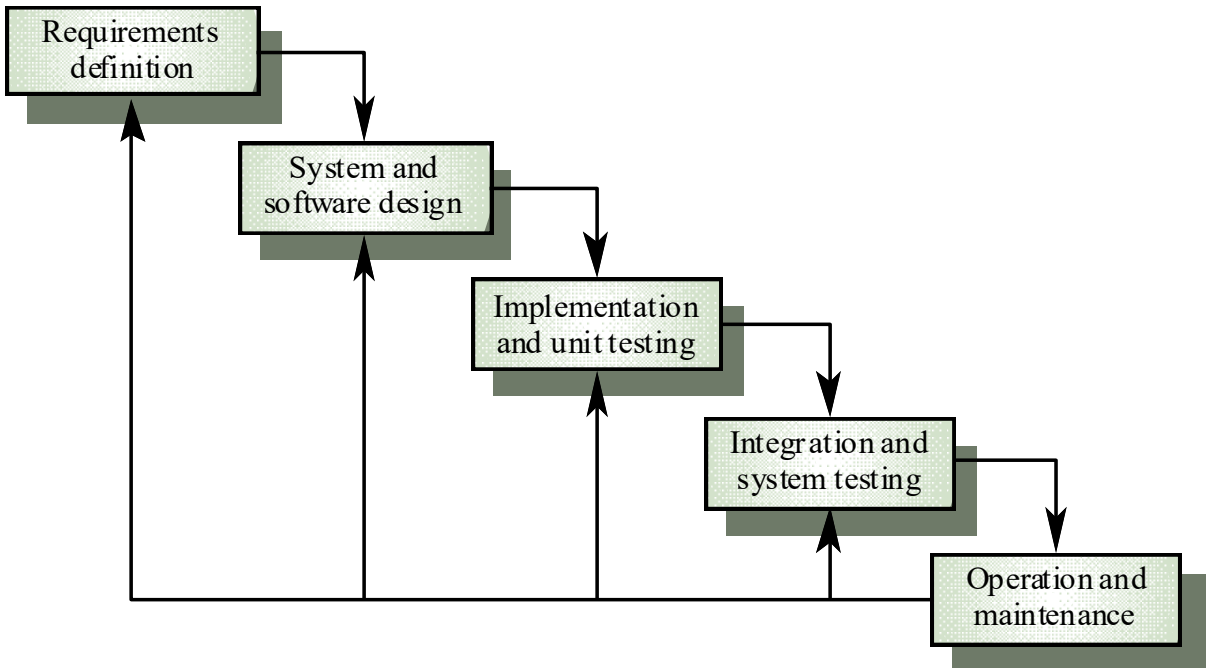
System and software design

Implementation and unit testing

Integration and system testing

Operation and maintenance

- The drawback of the waterfall model is the difficulty of accommodating change after the process is underway



Waterfall model problems:

- Real projects rarely follow it.
- Customer must have patience, not fast enough for delivery of modern internet based software.
- Each phase has to be completed before moving onto the next phase.
- It difficult to respond to changing customer requirements.
- Major mistake can be disastrous.
- Unnecessary delays.

Therefore, this model is only appropriate when the requirements are well-understood.

2. Prototyping:

Prototyping is a process that enables the developer to create a model of the software that must be built. The model can take one of three forms:

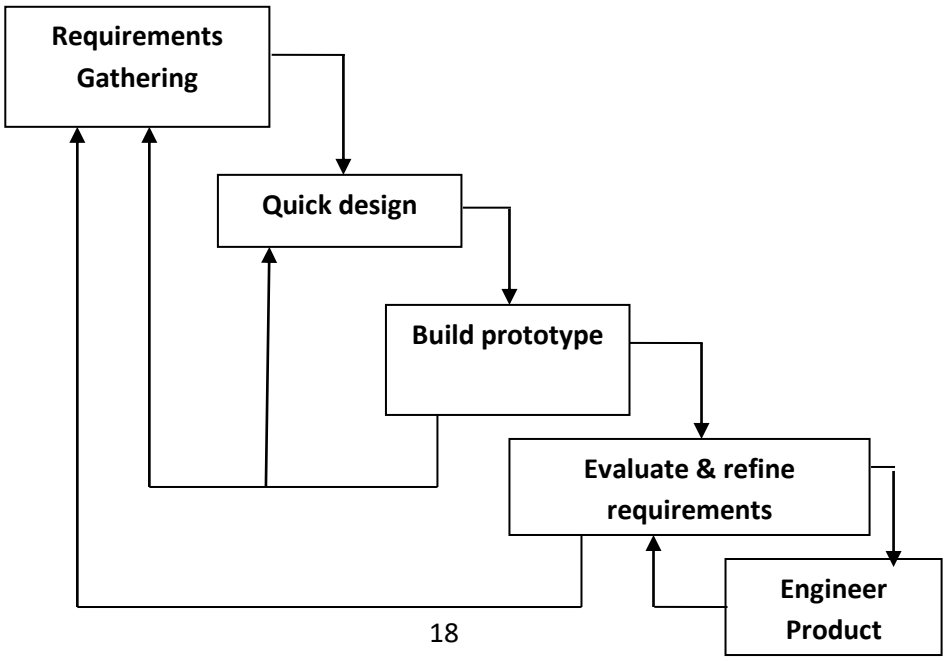
(1) a paper prototype that depicts human-machine interaction in a form that enables the user to understand how such interaction will occur.

(2) a working prototype that implements some subset of the function required of the desired software.

(3) an existing program that performs part or all of the function desired but has other features.

This model consists of:

- Gather requirements
- Developer & customer define overall objectives, identify areas needing more investigation – risky requirements
- Quick design focusing on what will be visible to user – input & output formats
- Use existing program fragments, program generators to throw together working version.
- Prototype evaluated and requirements refined.
- process iterated until customer & developer satisfied
    - then throw away prototype and rebuild system to high quality
    - alternatively can have evolutionary prototyping – start with well understood requirements



Prototyping Drawbacks:

- Customer may want to hang onto first version, may want a few fixes rather than rebuild. First version will have compromises.
- Developer may make implementation compromises to get prototype working quickly. Later on developer may become comfortable with compromises and forget why they are inappropriate.

3. Evolutionary development:

Exploratory development objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements and add new features as proposed by the customer.

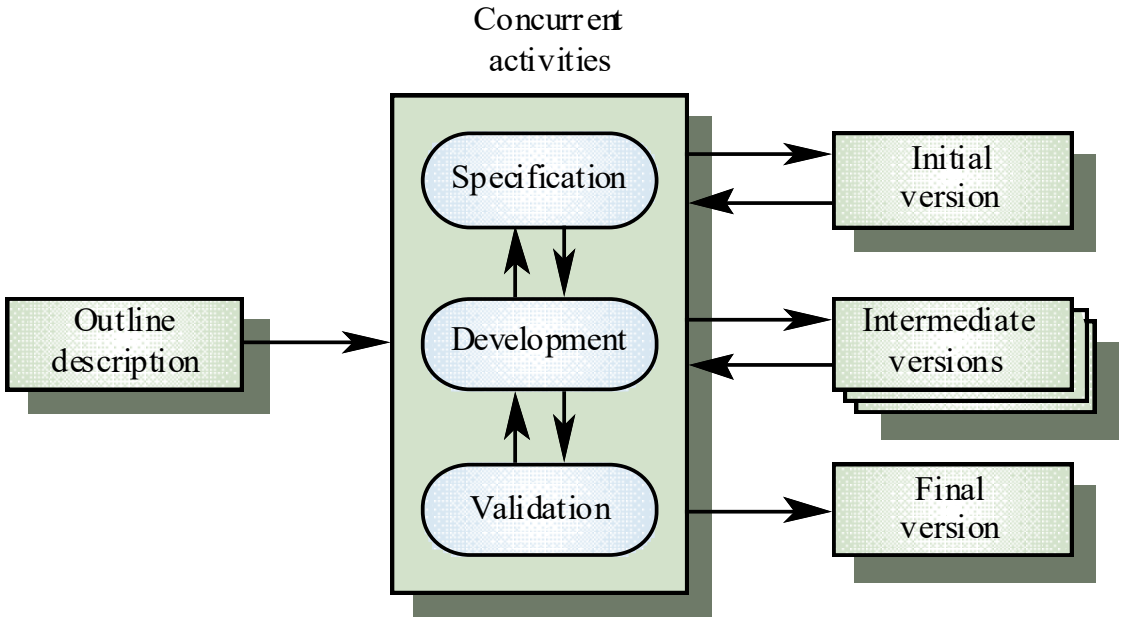
Throw-away prototyping objective is to understand the system requirements. Should start with poorly understood requirements to clarify what is really needed.

Applicability

- For small or medium-size interactive systems
- For parts of large systems (e.g. the user interface)
- For short-lifetime systems

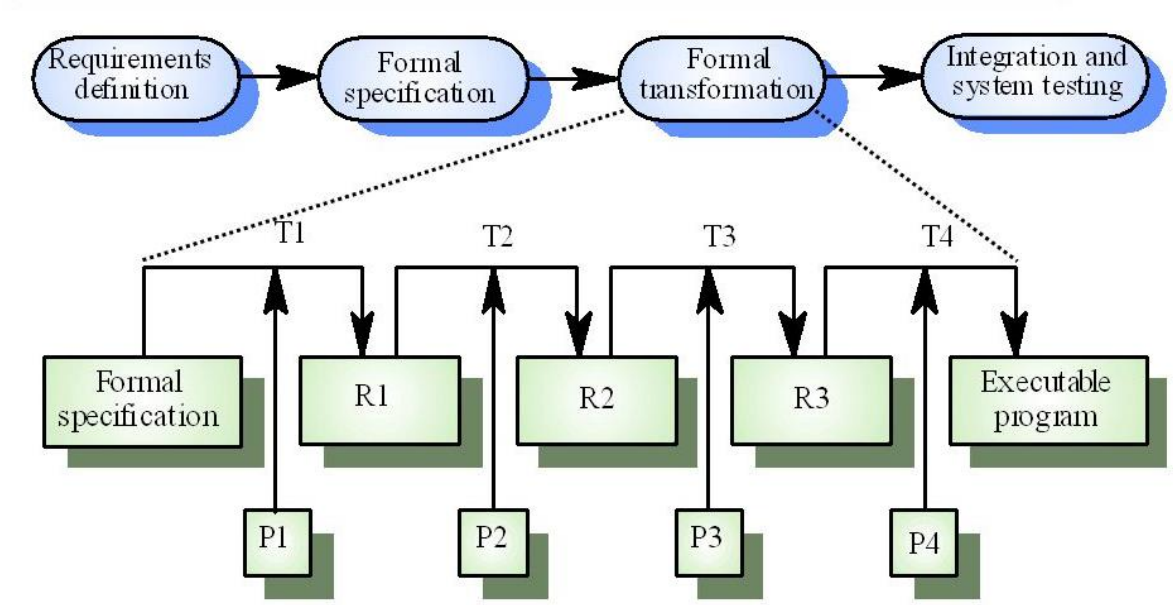
Problems

- Lack of process visibility
- Systems are often poorly structured
- Special skills (e.g. in languages for rapid prototyping) may be required



4. Formal systems development:

A mathematical model of a system specification is created. This model is then refined, using mathematical transformations that preserve its consistency, into executable code. Based on the transformation of a mathematical specification through different representations to an executable program. Transformations are 'correctness-preserving' so it is straightforward to show that the program conforms to its specification. This model presents both the theoretical basis for this and practical means of refinement from system specification to program code.



Proofs of transformation correctness

Applicability

- Critical systems especially those where a safety or security case must be made before the system is put into operation

Problems

- Need for specialised skills and training to apply the technique
- Difficult to formally specify some aspects of the system such as the user interface

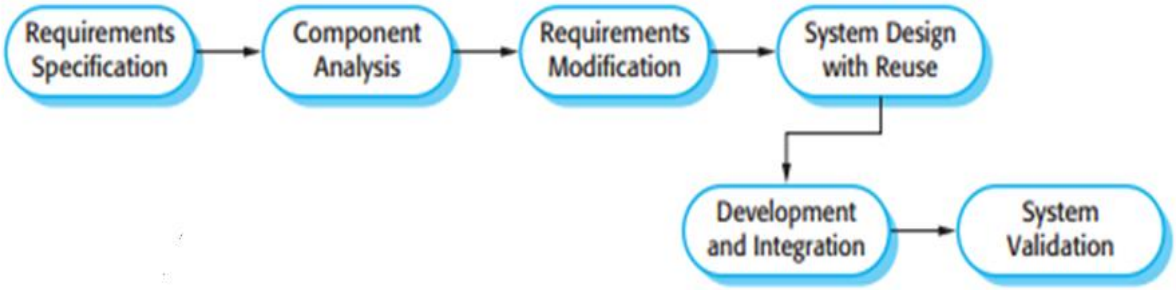
5. Reuse-Oriented Development ROD:

ROD is a method of software development in which a program is refined by producing a sequence of prototypes called models, each of which is automatically derived from the preceding one according to a sequence of defined rules. Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-

the-shelf) systems. However, in the 21st century, software development processes that focus on the reuse of existing software have become widely used.

Process stages:

a. Component analysis
b. Requirements modification
c. System design with reuse
d. Development and integration



This approach is becoming more important but still limited experience with it. There are three types of software component that may be used in a reuse-oriented process:

1. Web services that are developed according to service standards and which are available for remote invocation.
2. Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
3. Stand-alone software systems that are configured for use in a particular environment.

Benefits:

The reuse-oriented model can reduce the overall cost of software development compared with more tedious manual methods. It can also save time because each phase of the process builds on the previous phase which has already been refined. When carefully carried out, ROD can minimize the likelihood of errors or bugs making their way into the final product.

Limitations:

The reuse-oriented model is not always practical in its pure form because a full repertoire of reusable components may not be available. In such instances, some new program components must be designed. If not thoughtfully done, ROD can lead to compromises in perceived requirements, resulting in a product that does not fully meet the needs of its intended users.